ML-Based Gear Fault Diagnostic with Continuous Improvement

CS6301.503 Internet of Things -- Project 2

Ayush Bhardwaj | Patrick Dayton | Yatharth Singhal

1. Introduction

The objective of this project was to create a fault diagnostic and analysis tool similar to what is often used in industry to monitor physical systems. Reading data points continuously from multiple simulated inputs, our system uses an XGBoosted model and other machine learning techniques to classify faults in a simulated gear box. Originally trained in offline mode with a static dataset, the model performs continuous improvement by plowing new data back into the model. It classifies data and calculates diagnostics continuously. Diagnostic and analysis data sent to a visualization window to be read by a knowledgeable user.

This type of system is abundant in the world in almost all industries. Data might be ingested to monitor and diagnose data from the power grid, railroad lines, internet infrastructure, healthcare systems, and more. The world of IoT has decreased the price of sensors and data transfer allowing more and more data to be created for better monitoring the world around us..

2. Design & Implementation

We iterated through multiple technologies for several portions of the project before settling on our final, most accurate, design.

2.1 Data Generation and Storage

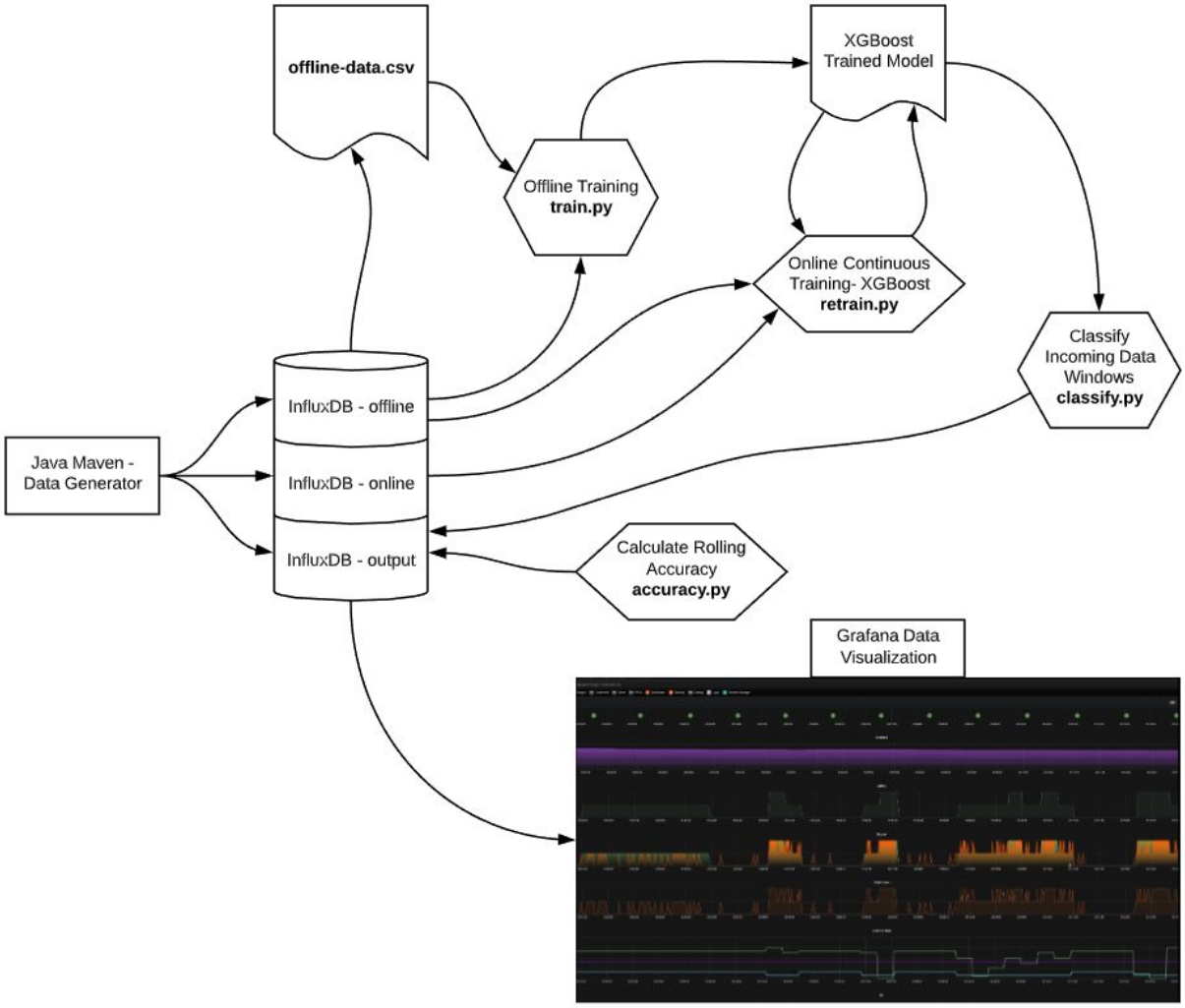
The generator was provided as a Maven (Java) project. It produces hundreds of metric readings per minute with four primary metrics with the labels *SR*, *Rate*, *GR*, & *Load*. Along with the four metrics, we receive a label of zero, one, or two where zero corresponds to nominal activity and one and two correspond to two types of fault.

Our first step was to spin up a local containerized version of the time series database InfluxDB using Docker and Docker Compose. Then we updated the data generator to port its data directly into the database.

An offline data set was produced by allowing the data generator to run continuously until we had more than 200 thousand readings. This was pulled from the time series database into a CSV format for faster loading during the development cycle. Through trial and error we found that we were able to boost accuracy by creating a data set from data generator with boosted error rates

(70% errors, 30% nominal) and using the --*online* flag in the Maven project when generating the data.

2.2 Machine Learning Model & Training



Initially our team used a deep neural net powered by Tensorflow to implement the system. Through testing we found that we were able to train faster and classify with higher accuracy by using an XGBoost model. XGBoost is a decision tree based ensemble model that uses gradient boosting principles to classify or regress. It's much more lightweight than Tensorflow and performs well in rapid classification and training settings.

2.3 Asynchronous Microservices - Training, Classification, and Metrics

Our system is built on a low-tech microservices based architecture where three different python scripts run asynchronously and loosely coupled. There are no direct interactions between the scripts and they all run continuously, pulling data from InfluxDB or sleeping if there is no data available.

We found that our model was most successful with a data window of 20 measurements. That is each microservice pulls the data that has been created since it's last iteration, splits the data into 20 measurement intervals, and applies itself to each of those intervals. Though 20 is the default it, and many other system settings, can be specified with command line flags and variables.

2.3.1 Classification Service

Each time this service runs, all data is since the previous iteration is retrieved via an offset that is tracked globally. All the queried data is split into data window sized vectors (default of 20 measurements) to be fed into the XGBoost model. The model gives one classification for each data window. This classification and the most occuring true label are saved to the output section of the database for metric visualization.

2.3.2 Retraining Service

The retraining service runs continuously, querying all the data since the previous iteration and concatenating it to the offline database. This database is then split into data window sized intervals to be trained upon. Once the training is complete, the model file is overwritten and the classifier service begins to use the newly trained model.

2.3.3 Accuracy Service

Each time the accuracy service runs, the previous 15 minutes of data from the output section of InfluxDB is queried. The total number of accurate data window predictions is divided by the total number of classifications in the past fifteen minutes to calculate a rolling 15 minute average of accuracy. This number is saved back to the output section of InfluxDB to be shown in the Metric Visualization system.

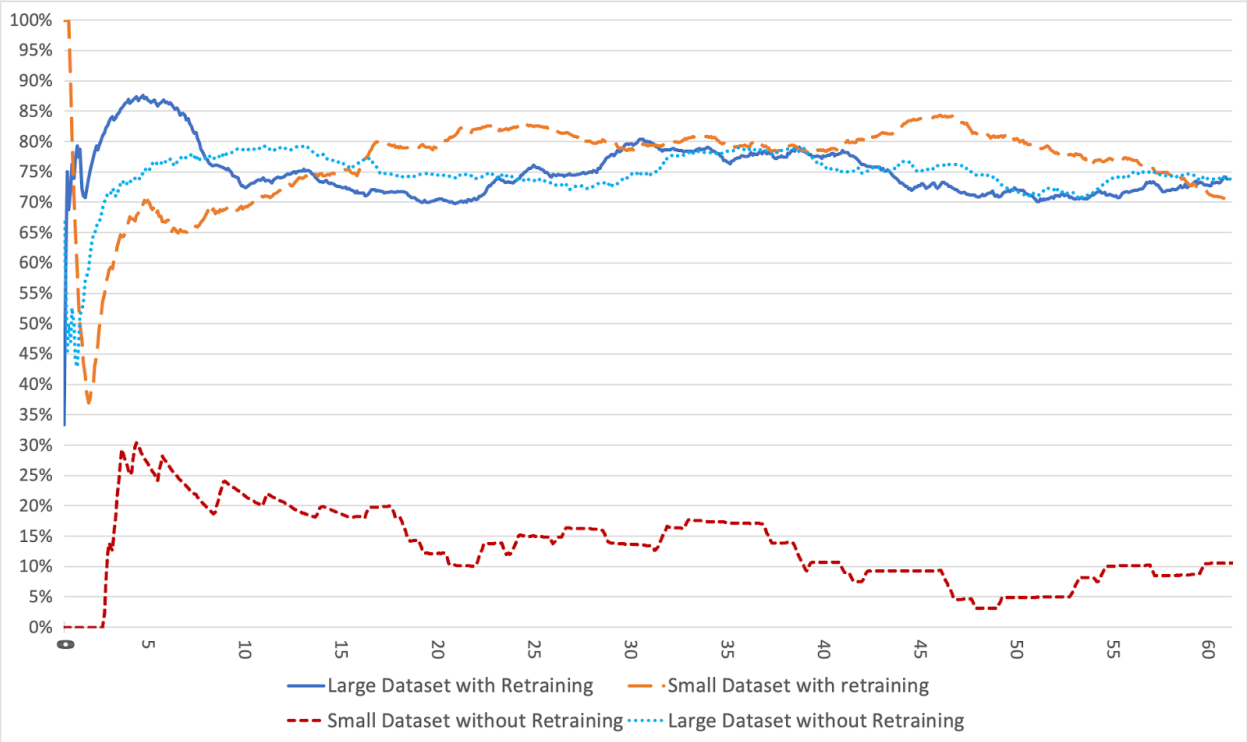
2.4 Metric Visualization

The free graphing software Grafana was used to show data outputs as it integrates seamlessly with InfluxDB and allows for real-time monitoring as one might require in a monitoring application such as this. We chose to run Grafana with Docker and Docker Compose. In it we

showed each of the metrics provided by the data generator (bottom two metric plots), our rolling accuracy number (purple), actual labels provided by the data generator (green shaded), predictions from our model (orange shaded), and a plot of when a new model was released by the online training service (green dots). An example of the visualization is shown below.

3. Results

Using our 15-minute rolling accuracy as our primary performance metric we ran four, one-hour tests. The default data window size of 20 measurements was used for all tests. We used two different offline training datasets, one small, one large consisting of 100 and 700,000 measurements respectively. The rolling fifteen minute accuracy average can be seen in the graph below:



The average accuracy for the final 45 minutes of run time for each test (after the rolling 15 minute average had time to run its course) were as follows:

- Large Dataset with Retraining - 74%
- Small Dataset with Retraining - 79%
- Small Dataset without Retraining - 12%
- Large Dataset without Retraining - 75%

Note that the one-hour testing length may not be long enough to rank the three better performing models with a high degree of statistical confidence. That being said, data suggests that in this particular model configuration, we can quickly reach a plateau of our near-maximum accuracy measurement with a relatively small amount of data. We see this since each of the three more

accurate models approached its average quickly after the system was enabled. A small training set though, understandable, yields poor results.

Along with seemingly better accuracy numbers, choosing to test with 70% errors allowed us to more evenly spread our time between each state of the system (zero, one, or two). Thus our accuracy did not get falsely inflated by often receiving 15 minutes of nominal behavior.

Furthermore, when comparing the classified plot with the labeled plot we can see that while not perfect, the classifier would be a useful tool in assisting a human eye in detecting anomalous behavior in a gear box.

4. Background & Contribution

Ayush Bhardwaj

- ➢ Researched for the possible Deep Neural Net for text classification.
- ➢ Researched upon weight initialisation and different activation functions for Neural Nets.
- ➢ Built initial Deep Neural Net in Tensorflow
- ➢ Created Offline Training datasets
- ➢ Worked upon increasing efficiency by testing different combinations of neurons, layers and Epochs.

Patrick Dayton

- ➢ Set up InfluxDB with Docker and Docker Compose for local time series data storage.
- ➢ Updated Java data generator Maven project to integrate with InfluxDB. Learned enough Java & Maven to get this working.
- ➢ Implemented Grafana to visualize metrics in real time.
- ➢ Created XGBoost model with train, classify, retrain, accuracy services
- ➢ YouTube demonstration of system

Yatharth Singhal

- ➢ Further developed the initial Neural Net made by the teammate.
- ➢ Testing the DNN for different values of Epochs, Activation Functions and learning rate and layers.
- ➢ Implemented saving and loading for DNN model.
- ➢ Implemented prediction and retraining the model for online learning.
- ➢ Researched about different types of classifiers and which one would be the most suitable, also tried setting up Docker and Docker compose.

5. Links

GitHub Repository: https://github.com/daytonpe/continuous-fault-diagnostic
YouTube Demo: https://www.youtube.com/watch?v=ANAbMNPWRVA&t=

6. Future Work

- ➢ Dockerize all of the microservices and run them all in Kubernetes.
- ➢ Perform more data engineering to smooth out the Machine Learning model and improve accuracy. This would include implementing discrete wavelet transform as a feature.
- ➢ Implement a faster and more accurate Tensorflow or Pytorch model which would likely outperform XGBoost with enough training.
- ➢ Further tune the XGBoost model for higher accuracy.

7. References

- ❖ https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d